Sweep as a Generic Pruning Technique

Nicolas Beldiceanu

SICS

Lägerhyddvägen 18
SE-75237 Uppsala, Sweden
Email: nicolas@sics.se

June 01 2000
SICS Technical Report T2000/08
ISSN 1100-3154
ISRN: SICS-T--2000/08-SE

Abstract This report presents a generic pruning technique that aggregates several constraints sharing some variables. The method is derived from an idea called sweep that is extensively used in computational geometry. A first benefit of this technique comes from the fact that it can be applied on several families of global constraints. A second main advantage is that it does not lead to any memory consumption problem since it only requires temporary memory that can be reclaimed after each invocation of the method. This technique has been used inside the SICStus finite domain solver in order to implement several global constraints.

Keywords Global constraint, sweep.

1 Introduction

The purpose of this report is to present a generic pruning technique for finite domain[1] constraint solving. This method is based on an idea that is widely used in computational geometry and that is called sweep [5, pages 10-11]. In dimension 2, a plane sweep algorithm solves a problem by moving a vertical line Δ from left to right[2]. The algorithm uses the two following data structures:

- one data structure called the sweep-line status, which contains some information related to the current position of the vertical line Δ,
- one data structure named the event point series, which holds the events to process, ordered in increasing order according to the abscissa.

The algorithm initializes the sweep-line status for the starting position of the vertical line Δ. Then the line Δ jumps from event to event; each event is handled and inserted or removed from the sweep-line status. One common application of the sweep algorithm is to solve the segments intersection problem [5, page 278] with a time complexity that depends both on the number of segments and on the number of segment intersections.

In our case, the sweep-line scans the values of a domain variable X we want to prune, and the sweep-line status contains a set of constraints that have to hold for that value of X where we currently are. The generic pruning technique, that we call *value sweep pruning*, accumulates the values to be currently removed from the domain of a variable Y that is different from the variable X we want to prune. If, for some position x of the sweep-line Δ, all values of Y have to be removed, then we will prune x from X. The method is based on the aggregation of several constraints that share some variables in common. Let:

- X and Y be two distinct domain variables,
- $C_1(V_{1,1},..,V_{1,n_1})..C_m(V_{m,1},..,V_{m,n_m})$ be a set of m constraints such that $\forall i \in 1..m : \{X,Y\} \subseteq \{V_{i,1},..,V_{i,n_i}\}$ (*i.e.* all constraints mention both variables X and Y).

The *value sweep pruning* algorithm will try to adjust the minimum[3] value of variable X according to the conjunction of the previous constraints by moving a vertical line Δ from the minimum value of X to its maximum value. In our case the events to process correspond to the start and to the end of forbidden 2-dimensional regions for the pair (X,Y) according to variable X.

---

[1] A domain variable is a variable that ranges over a finite set of integers; min(X), max(X), dom(X) and size_dom(X) denote respectively the minimum value, the maximum value, the set of possible values of variable X and the number of possible values for X.

[2] In general, a plane-sweep algorithm does not require, neither the sweep-line to be vertical, nor moving the sweep-line from left to right.

[3] It can also be used in order to adjust the maximum value, or even to prune completely the domain of a variable.

The next section presents forbidden regions, while the third section describes the *value sweep pruning* algorithm. The fourth section shows one other way of using *value sweep pruning*. Section 5 discusses in which context *value sweep pruning* can be applied and mentions three different examples where it was effectively used for implementing different kinds of non-overlapping constraints. Finally the last section shows how to enhance and extend *value sweep pruning* in different ways.

2 Forbidden regions

A forbidden region F according to one of the previous constraints C_i $(1 \le i \le m)$ and to two given variables X and Y, is defined by two intervals $inf_{F,X}..sup_{F,X}$ and $inf_{F,Y}..sup_{F,Y}$ such that $\forall x, y / x \in inf_{F,X}..sup_{F,X} \wedge y \in inf_{F,Y}..sup_{F,Y} : C_i(V_{i,1},..,V_{i,n_i})$ with the assignment $X = x$ and $Y = y$ has no solution. The next figure shows 5 constraints and their respective forbidden regions according to two given variables X and Y. The first constraint imposes X, Y and R to be pairwise distinct. The second and third constraints are usual arithmetic constraints. The fourth constraint can be interpreted as a non-overlapping constraint between two rectangles of respective origins X,Y and T,U and of respective sizes 2,4 and 3,2. Finally the last constraint is a parity constraint of the sum of X and Y.

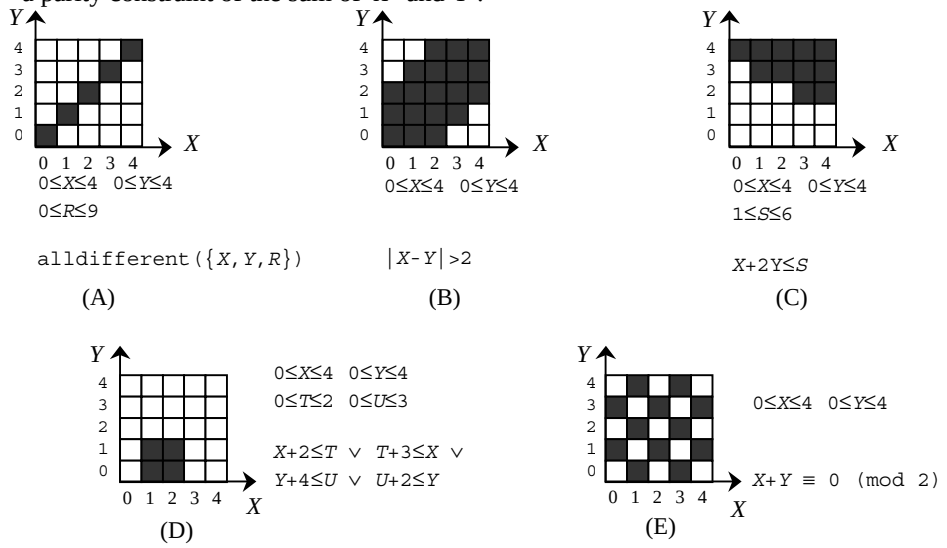


Fig. 1. Examples of forbidden regions

The *value sweep pruning* algorithm requires the forbidden regions of each constraint C_i $(1 \le i \le m)$ to be generated on request as a set of rectangles $R_{i,1},..,R_{i,n}$ such that:

- $R_{i,1} \cup .. \cup R_{i,n}$ represents all forbidden regions of constraint C_i according to variables X and Y. Note that we do not request the rectangles to form a partition of the forbidden space. This is because it sometime allows generating fewer rectangles: more than 3 rectangles are necessary to cover the forbidden regions of example B of Figure 1 if we would ask for a partition.
- $R_{i,1},..,R_{i,n}$ are sorted in increasing order on their respective start on the X axis.

This will be handled by providing the two following functions[4] for each triple (X,Y,C_i) we want to be used by the *value sweep algorithm*.

- get_first_forbidden_regions(X,Y,C_i), which correspond to:
  Let $first_{C_i}$ be the smallest value $\in \min(X)..\max(X)$ such that:

  $$\exists \text{ a forbidden region } R_{C_i} \text{ of } C_i \text{ with } \begin{cases} inf_{R_{C_i},X} \le first_{C_i} \le sup_{R_{C_i},X} \\ sup_{R_{C_i},Y} \ge \min(Y) \wedge inf_{R_{C_i},Y} \le \max(Y) \end{cases},$$

  Returns all forbidden regions R_{C_i} of C_i such that:

  $$\begin{cases} inf_{R_{C_i},X} \le first_{C_i} \le sup_{R_{C_i},X} \\ sup_{R_{C_i},Y} \ge \min(Y) \wedge inf_{R_{C_i},Y} \le \max(Y) \end{cases}.$$

- get_next_forbidden_regions$(X,Y,C_i,prev_i)$ ($prev_i$ is the position of the previous start-event of C_i), which correspond to:
  Let $next_{C_i}$ be the smallest value greater than $prev_i$ such that:

  $$\exists \text{ a forbidden region } R_{C_i} \text{ of } C_i \text{ with } \begin{cases} inf_{R_{C_i},X} = next_{C_i} \\ sup_{R_{C_i},Y} \ge \min(Y) \wedge inf_{R_{C_i},Y} \le \max(Y) \end{cases},$$

  Returns all forbidden regions R_{C_i} of C_i with:

  $$\begin{cases} inf_{R_{C_i},X} = next_{C_i} \\ sup_{R_{C_i},Y} \ge \min(Y) \wedge inf_{R_{C_i},Y} \le \max(Y) \end{cases}.$$

Finally, two additional functions check_if_in_forbidden_regions(x,y,C_i) and max_ysize_forbidden_regions(X,Y,C_i) have to be provided in order to respectively:

- test if given values $x \in \text{dom}(X)$ and $y \in \text{dom}(Y)$ belongs or not to a forbidden region of constraint C_i.
- compute an upper bound of the quantity $\max_{x \in \text{dom}(X)} |Forbid(Y,x)|$, where $Forbid(Y,x)$ is the set of values y of variable Y such that constraint $C_i(V_{i,1},..,V_{i,n_i})$ with the assignment $X = x$ and $Y = y$ has no solution.

---

[4] Two equivalent functions get_last_forbidden_regions and get_prev_forbidden_regions are also provided for the case where the sweep-line moves from the maximum to the minimum value.

If we consider constraint (E) of Figure 1 (*i.e.* $X+Y \equiv 0 \pmod 2$), and we assume $X \in 0..2$ and $Y \in 1..3$ then a complete scan of X would produce the following sequence of calls:

- get_first_forbidden_regions$(X, Y, X+Y \equiv 0 \pmod 2)$ returns regions $0..0,1..1$ and $0..0,3..3$,
- get_next_forbidden_regions$(X, Y, X+Y \equiv 0 \pmod 2, 0)$ returns region $1..1,2..2$,
- get_next_forbidden_regions$(X, Y, X+Y \equiv 0 \pmod 2, 1)$ returns regions $2..2,1..1$ and $2..2,3..3$.

A call to max_ysize_forbidden_regions$(X, Y, X+Y \equiv 0 \pmod 2)$ would return 2, since for any value of $X \in 0..2$ we have at most two forbidden positions for $Y \in 1..3$. We now show how to use the max_ysize_forbidden_regions(X, Y, C_i) function in order to get a condition, which tell us when *value sweep pruning* can for sure not bring anything. This condition can be used as a filter in order to avoid unnecessary calls to the *value sweep pruning algorithm*. Let:

- X and Y be two distinct domain variables,
- $C_1(V_{1,1},..,V_{1,n_1})..C_m(V_{m,1},..,V_{m,n_m})$ be a set of m constraints such that $\forall i \in 1..m : \{X, Y\} \subseteq \{V_{i,1},..,V_{i,n_i}\}$ (*i.e.* all constraints mention both variables X and Y).

If $\sum_{i \in 1..m} \text{max\_ysize\_forbidden\_regions}(X, Y, C_i) < \text{size\_dom}(Y)$ then *value sweep pruning* is not useful since all values of Y cannot be completely covered by the different forbidden regions.

3 The *value sweep pruning* algorithm

The *value sweep pruning* algorithm uses the following data structures:

- the sweep-line status contains the current possible values for variable Y according to the position of the vertical line Δ (*i.e.* the current position in X). More precisely the sweep status can be viewed as a cumulated profile (denoted P_{status}) that records for each position of Y the number of forbidden regions that currently intersect the sweep-line Δ. The basic operations required on this data structure are:
  - to initialize to empty the profile,
  - to add in the profile a task of height 1 that starts and ends at two given positions,
  - to remove from the profile a task of height 1 that starts and ends at two given positions,
  - to check if there exists a point of the profile that is situated at altitude 0,
  - to return a random point of the profile that is situated at altitude 0.
- the event point series (denoted Q_{event}) contains the start and the end+1 (+1 since the end is still forbidden), according to the X axis, of the forbidden regions

associated to the constraints $C_1,..,C_m$ and to variables X and Y. These events are sorted in increasing order. The basic operations required are:
- to initialize to empty the queue,
- to add an event in the queue,
- to extract an event of minimum value from the queue,
- To check if there exists at least one start-event associated to a specific constraint C_i $(1 \le i \le m)$.

This last primitive is the trigger that will be used in order to gradually insert the start and end-events associated to the forbidden regions of a given constraint C_i $(1 \le i \le m)$ when a start-event associated to constraint C_i is removed from the queue Q_{event}. Before going more into the detail of the algorithm, let us illustrate how it works on a concrete example. Assume we want to find out the minimum value of variable X according to the conjunction of the 5 constraints that were given in Figure 1. The next figure shows the status of the sweep-line Δ (*i.e.* the cumulated profile P_{status}) at the different positions of Δ on the X axis.

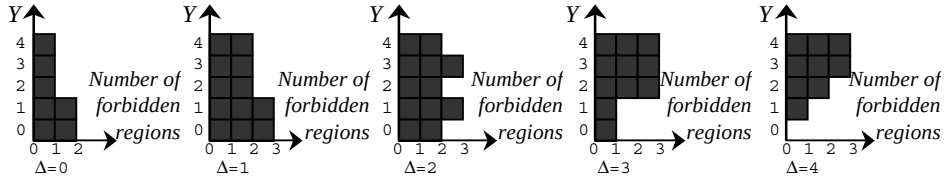


Fig. 2. Status of the sweep-line Δ at each stage of the algorithm

The minimum value of variable X is 4, since this is the first stage where the cumulated profile associated to the sweep-line has a point that is located at altitude 0 (*i.e.* at position 0 on the horizontal axis). We now give the main procedure.

Algorithm FINDMINIMUM($C_1,..,C_m$, X , Y)

Input. A set of constraints $C_1,..,C_m$ and two domain variables X and Y present in each constraint.

Output. An indication that no solution exists or the minimum value min_x of X , and a value $y \in \text{dom}(Y)$ such that (min_x, y) do not belong to any forbidden region associated to the constraints $C_1,..,C_m$ according to variables X and Y .

1. Initialize an empty event queue Q_{event} .
   for each constraint C_i $(1 \le i \le m)$ **do**
      for all forbidden regions R_{C_i} returned by get_first_forbidden_regions(X,Y,C_i) **do**
         Insert $\max(inf_{R_{C_i},X}, \min(X))$ in Q_{event} as a start-event.
         if $sup_{R_{C_i},X} + 1 \le \max(X)$ **then** Insert $sup_{R_{C_i},X} + 1$ in Q_{event} as an end-event.
   if Q_{event} is empty
   or the position m of the smallest event of Q_{event} is greater than $\min(X)$ **then**
      Set min_x to $\min(X)$, y to a random value of $\text{dom}(Y)$ and **exit**.
2. Initialize the sweep-line status to an empty profile P_{status} .
   for all non overlapping intervals $low..up / \forall v \in low..up : v \notin \text{dom}(Y)$ **do**

   $$\text{Insert rectangle } \begin{cases} origin : low \\ end : \quad up \\ high : \quad 1 \end{cases} \text{ in } P_{status}.$$

3. **while** Q_{event} is not empty **do**
4. Move the sweep-line Δ to the position m of the smallest event of Q_{event} .
   for each event E of Q_{event} that is at position m **do** HANDLEEVENT(E).
   if at least one point of the profile P_{status} is at altitude 0 **then**
      Set min_x to m , the current position of sweep-line Δ.
      Choose randomly an instant y where the height of the profile P_{status} is 0.
      exit.
5. **fail**.

The 5 steps of the algorithm respectively correspond to:

- the first step initializes the event queue Q_{event} to the start and end-events associated to the "earliest" forbidden regions associated to each constraint. Note that we only insert events that are effectively within $\min(X)..\max(X)$ and $\min(Y)..\max(Y)$. If no such events are found or if no event intersects $\min(X)$ then we exit from the procedure.
- the second step initializes the cumulated profile to 0 for the values that belong to $\text{dom}(Y)$ and to 1 otherwise. These last values will be forbidden forever, since no corresponding end-event was inserted in the event queue Q_{event} .

– the third step iterates on the positions of the sweep-line Δ.
– the fourth step extracts from the event queue Q_{event} all events associated to the current position of the sweep-line Δ, and handle them. After, it checks if there exist at least one feasible solution for the current position of Δ and exit from the procedure if it is effectively the case.
– finally the last step returns a failure since the sweep-line Δ did a complete scanning of the domain of variable X without finding any solution.

Holes in the domain of variable X are handled in the same way as constraints $C_1,..,C_m$: one additional constraint that, for each interval of consecutive removed values, generates start and end-event. The next procedure specifies how to handle start and end-event points.

Algorithm HANDLEEVENT(E)

1. Extract E from Q_{event}.

   Get the corresponding forbidden region R_E and constraint C_E.

   if E is a start-event **then** Insert rectangle $\begin{cases} origin: \max\left(inf_{R_E,Y}, \min(Y)\right) \\ end: \quad \min\left(sup_{R_E,Y}, \max(Y)\right) \\ high: \qquad 1 \end{cases}$ in P_{status}.

      if Q_{event} do not contain any start-event associated to constraint C_E **then**
         Let $prev_E = \max\left(inf_{R_E,X}, \min(X)\right)$.
         for all forbidden regions R_{C_E} returned by
                  get_next_forbidden_regions$\left(X, Y, C_E, prev_E\right)$ **do**
            Insert $inf_{R_{C_E},X}$ in Q_{event} as a start-event.
            if $sup_{R_{C_E},X} + 1 \le \max(X)$
            then insert $sup_{R_{C_E},X} + 1$ in Q_{event} as an end-event.

   else Remove rectangle $\begin{cases} origin: \max\left(inf_{R_E,Y}, \min(Y)\right) \\ end: \quad \min\left(sup_{R_E,Y}, \max(Y)\right) \\ high: \qquad 1 \end{cases}$ from P_{status}.

According to the fact that we have a start or an end-event E, we add or subtract a rectangle of height 1 to the cumulated profile P_{status}. The extremities of this rectangle match respectively the start and the end on the Y axis of the forbidden region that is associated to the event E. When E was the last start-event of a given constraint C_E, we search for the next events of C_E and insert them in the event queue Q_{event}.

One reason for returning a random feasible solution is that it will be used as a first check in order to avoid running again systematically the complete algorithm if the previous returned feasible solution is still feasible[5]. The motivation to return a random

[5] For this check, we will use the check_if_in_forbidden_regions function that was introduced in section 2.

value comes from the fact that, if we use the *value sweep pruning* algorithm for pruning several variables, we don't want to get the same feasible solution for several variables, since one single future assignment could invalidate this feasible solution. This would result in reevaluating again the *value sweep pruning* algorithm for several variables.

If we use the *value sweep pruning* algorithm for doing a complete pruning then each call to the algorithm will lead to a complete sweep over the domain of the variable we want to prune. However, if we use the algorithm for adjusting the minimum or maximum value then we will have a complete sweep on each branch of the search tree. This is because the sweep process will be stopped each time we find a first feasible solution.

Let us assume the total number of forbidden regions intersecting the initial domain of variables X and Y to be denoted *nforbid* . For a complete sweep, the next table indicates the number of times each basic operation will be respectively performed.

Table 1. Maximum number of calls to each basic operation

Basic operation	Maximum number of calls
Initialize to empty the queue	1
Add an event in the queue	2 x *nforbid*
Extract an event of minimum value from the queue	2 x *nforbid*
Check if there exists at least one start-event associated to a specific constraint	*nforbid*
Initialize to empty the profile	1
Add in the profile a task	*nforbid*
Remove from the profile a task	*nforbid*
Check if there exists a point of the profile that is situated at altitude 0	2 x *nforbid*
Return a random point of the profile that is situated at altitude 0	1

Note that from a deductive point of view, the *value sweep pruning* algorithm is similar to the work done in [9]. However the main difference is that the set of forbidden regions associated to each pair of variables (X,Y) was stored explicitly in a quadtree for which one need to restore the previous state on backtracking. With the sweep, one can reclaim the data structures (*i.e.* the queue and the profile) after each invocation to the method.

4 Using the *value sweep pruning* algorithm for adjusting *decreasing monotonic* domain variables

This paragraph explain another way of using the *value sweep pruning* algorithm for adjusting the maximum value of a certain class of domain variables that we introduce in the next definition.

Definition *decreasing monotonic domain variable according to a given constraint*
A domain variable V_i $(1 \leq i \leq n)$ is called *decreasing monotonic* according to a constraint $C(V_1,..,V_n)$ if the following condition holds: $C(v_1,..,v_i,..,v_n)$ true $\Rightarrow$ $\forall v < v_i : C(v_1,..,v,..,v_n)$ true .

We now state the conditions under which *value sweep pruning* can be used for adjusting the maximum value of a *decreasing monotonic* variable. Let:

- X , Y and L be 3 distinct domain variables,
- $C_1(V_{1,1},..,V_{1,n_1})..C_m(V_{m,1},..,V_{m,n_m})$ be a set of m constraints such that $\forall i \in 1..m : \{X,Y,L\} \subseteq \{V_{i,1},..,V_{i,n_i}\}$ (*i.e.* all constraints mention both variables X , Y and L),
- L is a *decreasing monotonic* variable according to each constraint of $C_1(V_{1,1},..,V_{1,n_1})..C_m(V_{m,1},..,V_{m,n_m})$.

We compute the maximum value of variable L by using a binary search, where at each step, we call the *value sweep pruning* algorithm in order to check for feasibility. Let us show a simple example where we have the two following non-overlapping constraints C_1 , C_2 , corresponding to the pairs of rectangles (R,R1), (R,R2) respectively, and the two following latest-end constraints C_3 , C_4 :

$C_1 : X + L \leq 3 \vee 4 \leq X \vee Y + 3 \leq 2 \vee 4 \leq Y$,

$C_2 : X + L \leq 5 \vee 7 \leq X \vee Y + 3 \leq 4 \vee 5 \leq Y$,

$C_3 : X + L \leq 9$,

$C_4 : Y + 3 \leq 7$.

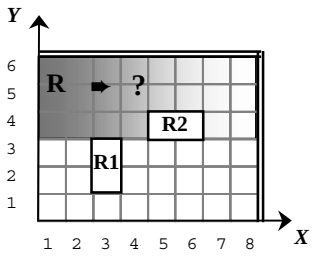


Fig. 3. Finding the maximum length of R

R is a rectangle beginning at (X,Y) with a width of L and a height of 3. X , Y and L are domain variables with respective domain 1..8, 1..4 and 1..8. From the previous definition, L is a decreasing monotonic variable according to constraints C_1 and C_2 . So, we evaluate the maximum value of L by searching for the largest value l such that the *value sweep pruning* algorithm does not return a failure (*i.e.* finds a feasible solution). The next figure shows the sequence of tries made by the binary search in order to find out that 5 is the maximum feasible value for variable L . For each try, we indicate the two forbidden regions F1 and F2 respectively associated to

rectangles R1 and R2. For try (A) where we get a positive answer, we also show a feasible solution for placing rectangle R with an effective length of 5. In this solution, R starts at coordinates $(4,1)$. Since the two forbidden regions overlap, we given them explicitly as a quadruple (FX, FY, LX, LY) where FX is the X coordinate of the leftmost point of the forbidden region, FY is the Y coordinate of the lowest point of the forbidden region, LX and LY are the size of the forbidden region on the X and Y axes. The dashed lines on the X and Y axes correspond to values that are removed by the latest-end constraints C_3, C_4 according to the sizes L,3 of rectangle R on the X and Y axes.

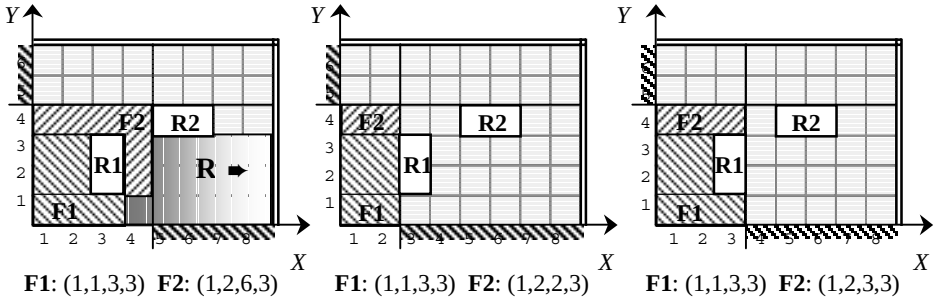


Fig. 4. The different calls to the *value sweep pruning* algorithm for localizing the maximum length of rectangle R

5 Structure of the global constraints for which one can apply *value sweep pruning*

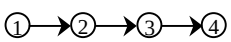
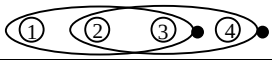
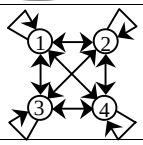
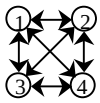
Although *value sweep pruning* can be applied to any arbitrary set of constraints for which one provide the forbidden regions, this may not be very easy to use in practice in a systematic way for at least two reasons:

- in a system where no global constraints are used, one would have to identify automatically, within the set of all posted elementary constraints, the pertinent subsets of constraints for which the method can be applied. This would cause a similar problem as extracting automatically specific structures from a set of linear constraints for linear programming.
- in a system where one would use ad-hoc global constraints, the method would be useless, unless one provides a specific implementation of *value sweep pruning* for each global constraint.

The purpose of this section is to characterize the applicability of *value sweep pruning* according to the classification of the global constraints we described in [1]. Before presenting typical global constraint patterns where one can apply *value sweep pruning*, we first shortly recall the main principles of the classification.

Global constraints are defined in terms of graph properties that have to hold, where the graph corresponds to a structured network of the same elementary constraints. The arcs are defined with a set of predefined arc generators that correspond to classical regular structures one can find in the graph literature [8, pages 140-153]. The next table gives examples of regular structures that we provide. We use the following pictogram for the graphical representation of a constraint network. We employ an arrow for arity 2, and an ellipse for a bigger arity. In these last cases, since the vertices of an arc are ordered, a circle at one of the extremity of the ellipse indicates the "direction" of the ellipse. For example, the ellipse that contains vertices 1, 2 and 3 means that a 3-ary constraint is applied on nodes 1, 2, and 3 in this specific order.

Table 2. Examples of regular structures produced by the arc generator

ARC GENERATOR	ARC ARITY	EXAMPLE
PATH	2	
PATH	3	
CLIQUE	2	
CLIQUE(≠)	2	

The graph properties that one wants to be verified on the final graph (*i.e.* the graph from which we remove the arcs associated to constraints that do not hold) are constraints on usual graph characteristics such as:

- NVERTEX the number of vertices of the final graph,
- NARC the number of arcs of the final graph,
- NSCC the number of strongly connected components of the final graph,
- MAX_NSCC the number of vertices of the largest strongly connected component of the final graph.

For instance, the number of distinct values constraint $\text{nvalue}(D, \{V_1,..,V_n\})$ where $\{V_1,..,V_n\}$ is a collection of domain variables and D is a domain variable that is equal to the number of distinct values of $\{V_1,..,V_n\}$, is defined in terms of graph property by:

- ARC GENERATOR : CLIQUE,
- ARC ARITY : 2,
- ARC CONSTRAINT: =,
- GRAPH PROPERTY: NSCC = D.

Part (A) of next figure shows the initial graph associated to the constraint $\text{nvalue}(D, \{V_1, V_2, V_3, V_4\})$, while part (B) gives the final graph associated to the solution $\text{nvalue}(2, \{8, 3, 8, 8\})$. The constraint holds since D is equal to the number

of strongly connected components of the final graph where we only keep the arc constraints that hold.

Fig. 5. The number of distinct value constraint

Next table shows 3 global constraint patterns where *value sweep pruning* can be applied.

Table 3. Patterns of global constraints for *value sweep pruning*

	PATTERN 1	PATTERN 2	PATTERN 3
ARC GENERATOR	CLIQUE	CLIQUE(≠)	PATH
ARC ARITY	2	2	n (n>2)
ARC CONSTRAINT PROPERTY	reflexive symmetric		
GRAPH PROPERTY	MAX_NSCC≤1	NARC=|VERTEX|^2-|VERTEX|	NARC=|VERTEX|-n+1

We now discuss each pattern separately.

- Since the first pattern imposes the final graph to have strongly connected components with no more than one node, such that each node has one loop, we have to enforce the negation of all constraints between a node and the other nodes. This will be the set of constraints we consider in the *value sweep pruning* algorithm. For the negation of the arc constraint we will have to provide the set of functions introduced in section 2 for defining the corresponding forbidden regions. One typical instance of the first pattern is the $\text{alldifferent}(X_1 - Y_1, .., X_n - Y_n)$ constraint that enforces all differences to be pairwise distinct. In this case, the associated arc constraint is $X_i - Y_i = X_j - Y_j$.
- The second pattern enforces all arc constraints to hold. For this reason we will consider for the *value sweep pruning* algorithm the constraints between a node and the other nodes. We will have to provide the forbidden regions associated to the given arc constraint.
- The third pattern describes a sliding constraint on group of consecutives variables. In this case we consider each pair of consecutive variables and all the arc constraints that mention that pair of variables to be the set of constraints that is given to the *value sweep pruning* algorithm.

The *value sweep pruning* algorithm was used in order to implement within the SICStus finite domain solver [2] 3 global constraints that correspond to the second pattern. These global constraints have respectively the following arc-constraints:

- The non-overlapping constraint between rectangles that is defined in [1, page 14],

– The cyclic non-overlapping constraint between rectangles that is defined in [1, page 14],
– The minimum distance constraint between rectangles, where the minimum distance depends of the type of the rectangles; a matrix gives for each pair of possible types the corresponding minimum distance.

The *value sweep pruning* algorithm was implemented once and different functions were given for defining the forbidden regions associated to the 3 previous arc-constraints.

6 Variants of *value sweep pruning*

6.1 Making *value sweep pruning* stronger

In the standard *value sweep pruning* algorithm, all forbidden regions are derived from individual elementary constraints. However it is possible to get bigger forbidden regions from the conjunction of several elementary constraints or from a given global constraint. We illustrate this last point by the following example.

Let us assume we have the constraint $\text{alldifferent}(X_1 - Y_1,..,X_n - Y_n)$ that enforces all differences to be pairwise distinct. Using a pruning algorithm [3], [6] that finds out all edges between a pair of variables $X_i, Y_i \ \ (1 \le i \le n)$ and the pairs of values x, y that do not belong to any maximum matching of size n, one can generate forbidden regions that can be accumulated by our *value sweep pruning* algorithm. Note that when these forbidden pairs of values are considered separately, they are usually not directly useful for reducing the domains of the variables.

6.2 Disjunctive *value sweep pruning*

This paragraph introduces a variant that is useful for integrating disjunctive constraints within the *value sweep pruning* algorithm. Let:

– X , Y , $Z_1,..,Z_d$ be $2+d$ distinct domain variables,
– $C_1(V_{1,1},..,V_{1,n_1})..\,C_m(V_{m,1},..,V_{m,n_m})$ be a set of m constraints such that: $\forall i \in 1..m : \{X, Y, Z_1,..,Z_d\} \subseteq \{V_{i,1},..,V_{i,n_i}\}$ (*i.e.* all constraints mention all variables X , Y and $Z_1,..,Z_d$).

Such that variables $Z_1,..,Z_d$ have to satisfy one of the two following constraints $CZ_1(Z_1,..,Z_d)$ or $CZ_2(Z_1,..,Z_d)$.

The standard *value sweep pruning* algorithm will not catch the disjunctive constraint. In order to handle actively the disjunctive constraint, we have to run two versions of the *value sweep pruning* algorithm. Each version will enforce one alternative of the disjunction which may lead restricting the domain of variables

$Z_1,..,Z_d$ and thus result in bigger forbidden regions for variables X and Y. The process will be stopped as soon as one of the two sweep-lines finds a feasible solution. A typical example of disjunctive *value sweep pruning* is when the two constraints $CZ_1(Z_1,..,Z_d)$ and $CZ_2(Z_1,..,Z_d)$ correspond to two possible sets of values for the attributes of an object [4, page 10].

6.3 Multi-dimensional *value sweep pruning*

It is possible to generalize the *value sweep pruning* algorithm to more than two variables in order to consider multi-dimensional forbidden regions. We first come up with a generalized definition of *value sweep pruning* and then discuss how the algorithm is affected. Let:

- $X_1,..,X_d$ be d distinct domain variables,
- $C_1(V_{1,1},..,V_{1,n_1})..C_m(V_{m,1},..,V_{m,n_m})$ be a set of m constraints such that $\forall i \in 1..m : \{X_1,..,X_d\} \subseteq \{V_{i,1},..,V_{i,n_i}\}$ (*i.e.* all constraints mention all variables $X_1,..,X_d$).

The generalized *value sweep pruning* algorithm will try to adjust the minimum[6] value of variable X_1 according to the conjunction of the previous constraints by moving a vertical line Δ from the minimum value of X_1 to its maximum value. In our case the events to process correspond to the start and to the end of forbidden d-dimensional regions for $X_1,..,X_d$ according to variable X_1.

From an algorithmic point of view the only thing to modify is to generalize the cumulated profile of the sweep status to a quadtree or an octree [7], which stores the number of forbidden regions that overlap one point associated to a value of X_1 and to variables $X_2,..,X_d$.

7 Conclusion

We have presented a *value sweep pruning* algorithm that performs global constraint propagation by aggregating several constraints that share two variables in common. This method is quite general and can be applied on a wide range of constraints. The usual way to handle finite domain constraints is to accumulate forbidden one-dimensional regions in the domain of the variables of the problem. However this is inefficient for constraints that do not initially have any forbidden one-dimensional forbidden regions since they have to be handled in a "generate and test" way (*i.e.* forbidden values appears only after fixing some variables). *Value sweep pruning* is an alternative that allows accumulating forbidden regions much more early in time. One

[6] As for the 2-dimensional case, it can also be used in order to adjust the maximum value, or even to prune completely the domain of a variable.

key point is that we do not represent explicitly all forbidden regions but rather ask for them in a focused way in order to try to perform specific pruning.

Acknowledgements

Thanks to Mats Carlsson for useful comments on an earlier draft of this report and for implementing the *value sweep pruning* algorithm.

References

1. Beldiceanu, N.: Global Constraints as Graph Properties on Structured Network of Elementary Constraints of the Same Type. SICS Technical Report T2000/01, (2000).
2. Carlsson M., Ottosson G., Carlson B.: An Open-Ended Finite Domain Constraint Solver. Proc. Programming Languages: Implementations, Logics, and Programs, (1997).
3. Costa, M-C.: Persistency in maximum cardinality bipartite matchings. Operation Research Letters 15, 143-149, (1994).
4. Herroelen, W., Demeulemeester, E., De Reyck, B.: A Classification Scheme for Project Scheduling Problems. in: Weglarz J. (Ed.), Handbook on Recent advances in Project Scheduling, Kluwer Academic Publishers, (1998).
5. Preparata, F. P., Shamos, M. I.: Computational geometry. An introduction. Springer-Verlag, (1985).
6. Régin, J-C.: A filtering algorithm for constraints of difference in CSP. In Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 362-367, (1994).
7. Samet, H: The Design and Analysis of Spatial Data Structures. Addison-Wesley, (1989).
8. Skiena, S.: Implementing Discrete Mathematics. Combinatoric and Graph Theory with Mathematica. Addison-Wesley, (1990).
9. du Verdier, F. R.: Résolution de problèmes d'aménagement spatial fondée sur la satisfaction de contraintes. Validation sur l'implantation d'équipements électroniques hyperfréquences. PhD. thesis of University Claude Bernard-Lyon I, (July 1992). In French.